A Protocol for the Atomic Capture of Multiple Molecules on Large Scale Platforms

Marin Bertier, Marko Obrovac, and Cédric Tedeschi

IRISA / INRIA, France
firstname.lastname@inria.fr

Abstract. With the rise of service-oriented computing, applications are more and more based on coordination of autonomous services. Envisioned over largely distributed and highly dynamic platforms, expressing this coordination calls for alternative programming models. The chemical programming paradigm, which models applications as chemical solutions where molecules representing digital entities involved in the computation, react together to produce a result, has been recently shown to provide the needed abstractions for autonomic coordination of services.
However, the execution of such programs over large scale platforms raises several problems hindering this paradigm to be actually leveraged. Among them, the atomic capture of molecules participating in concurrent reactions is one of the most significant.
In this paper, we propose a protocol for the atomic capture of these molecules distributed and evolving over a large scale platform. As the density of possible reactions is crucial for the liveness and efficiency of such a capture, the protocol proposed is made up of two sub-protocols, each of them aimed at addressing different levels of densities of potential reactions in the solution. While the decision to choose one or the other is local to each node participating in a program's execution, a global coherent behaviour is obtained. Proof of liveness, as well as intensive simulation results showing the efficiency and limited overhead of the protocol are given.

1 Introduction

With the widespread adoption of the Service-Oriented Architecture (SOA) paradigm, large scale computing platforms have recently undergone a new shift in their shape and usage. Within such platforms, the basic entity is a *service*, *i.e.*, an encapsulation of some computing, storage, or sensor device, to be used by users or combined with other services. On top of these platforms, applications now commonly compose these services dynamically, under the shape of *workflows*, *i.e.*, temporal compositions of services. To run over emerging highly distributed and dynamic platforms, without any central authority or orchestrator, services need to be able to coordinate themselves autonomously, in a fully-distributed fashion. In this context, programming models need to be rethought in order to provide the right abstractions for this coordination, while taking into account the distribution and dynamics of the underlying platform.

Artificial chemistries [1], which are chemically-inspired information processing models, have regained *momentum* in this context, and are now used to model this *ecosystem* of services [2]. More concretely, the chemical programming paradigm, initially developed to write highly parallel programs, was identified to provide the right level of abstraction for this context [3]. Within the basic version of the chemical programming model [4], a program is envisioned as a *chemical solution* where molecules of data float and react according to some *reaction rules* specifying the program, to produce new data (products of reactions). At runtime, reactions arise in an implicitly autonomous and parallel mode, and in a non-deterministic order. When no more reactions are possible, the solution contains the result of the computation.

While the chemical paradigm allows the easy design of coordination protocols, running these chemical specifications over distributed platforms is still a widely open issue. Among one of the most significant barriers to be lifted is the atomic capture of multiple molecules satisfying a reaction. At runtime, a molecule can potentially participate in several concurrent reactions. However, it is allowed to participate in only one. Otherwise, the logic of the program would be broken. This problem is exemplified in Section 2.

Let us slightly refine the problem considered in this paper. We consider a chemical program made of a multiset of data, and of a set of rules acting concurrently on them. Both data and rules are distributed amongst a set of nodes on which the program runs. Each node periodically tries to fetch molecules needed for their assigned reactions. As several molecules can satisfy the pattern and conditions of several reactions performed concurrently by different nodes, the same molecule can be requested by several nodes at the same time, inevitably leading to conflicts. Mutual exclusion on the molecules is thus mandatory.

Although our problem resembles the classic resource allocation problem [5], it differs in several aspects. Firstly, the molecules are exchangeable to some extent. Molecules requested must match a pattern defined in the reaction rule a node wants to perform. In other words, we differentiate two processes which are i) finding molecules matching a pattern (achieved by a *discovery protocol*), and ii) obtaining them to perform reactions (achieved by a *capture protocol*).

Secondly — and following the previous point — the platform envisioned is at large scale, and the resources dispatched over the nodes are dynamic: molecules are deleted when they react, and new ones are created. Thus, the protocol to discover molecules should be scalable and dynamic. Likewise, the number of resources/molecules (and possible reactions) will fluctuate over time, influencing the design of the capture protocol. Bear in mind that once the holder of a matching molecule is located, the scale of the network is of less importance, since only the requester and holder of the molecule are involved in the capture protocol.

Finally, and to sum up, our objective is to define a protocol for the atomic capture of multiple molecules, that dynamically and efficiently adapts to the density of potential reactions in the system.

Contribution. Our contribution is a distributed protocol mixing two sub-protocols inspired by previous works on distributed resource allocation, and adapted to the distributed runtime of *chemical* programs.

The first sub-protocol, referred to as the *optimistic* one, assumes that the number of molecules satisfying some reaction's pattern and condition is high, so only few conflicts for molecules will arise, nodes being likely to be able to grab distinct sets of molecules. While this protocol is simple, fast, and has a limited communication overhead, it does not ensure liveness when the number of conflicts increases. The second one, called *pessimistic*, slower, and more costly in terms of communication, ensures liveness in presence of an arbitrary number of conflicts. Switching from one protocol to the other is achieved in a scalable, distributed fashion, based on local success histories in grabbing molecules. A proof of liveness of our protocol is given, and its efficiency is discussed through a set of simulation results. Note that this work, to our knowledge, pioneers research on the distributed execution of *chemical* programs.

Organisation of the paper. The next section presents the chemical programming paradigm in more details, highlights the need for the atomic capture and describes the system model used throughout the paper. Section 3 details the sub-protocols, their coexistence, and the switch from one to the other one. Proofs of liveness and fairness are also given for the complete protocol. Section 4 presents the simulation results and discusses the efficiency and overhead of the protocol. Related works are presented in Section 5. Section 6 concludes.

2 Preliminaries

Different systems require different algorithms for performing atomic operations varying in complexity. This section describes the programming and system models which compose the required conditions for the protocol proposed.

2.1 Chemical Programming Model

The chemical model was initially proposed for a natural expression of parallel processing, by removing artificial structuring and serialisation of programs, focusing only on the problem logic. Following the chemical analogy, data are molecules floating in a solution. They are consumed according to some reaction rules, *i.e.*, the program, producing new molecules, *i.e.*, resulting data. These reactions take place in an implicitly parallel and autonomous way, until no more reactions are possible, a state referred to as *inertia*. This model was first formalised by GAMMA [4], in which the solution is a multiset of molecules, and reactions are rewriting rules on this multiset. A rule **replace** P **by** M **if** V consumes a set of molecules N satisfying the pattern P and the condition V, and produces a set of molecules M. We want to emphasise here that consumption is the only possible change of state a molecule can be subjected to: once it has been consumed, it vanishes from the multiset completely, meaning molecules are only

created and deleted, never updated nor recreated. For the sake of illustration, let us consider the following chemical program made up of two rules applied on a multiset of strings, that counts the aggregated number of characters in words with more than two letters:

$$\begin{array}{l}\textbf{let } \mathit{count} = \textbf{ replace } s :: \mathit{string} \textbf{ by } \mathit{len}(s) \textbf{ if } \mathit{len}(s) >= 2 \textbf{ in} \\ \textbf{let } \mathit{aggregate} = \textbf{ replace } x :: \mathit{int}, y :: \mathit{int} \textbf{ by } x + y \textbf{ in} \\ \quad \langle\ \text{"}\mathit{maecenas}\text{"}, \text{"}\mathit{ligula}\text{"}, \text{"}\mathit{massa}\text{"}, \text{"}\mathit{varius}\text{"}, \text{"}\mathit{a}\text{"}, \text{"}\mathit{semper}\text{"} \\ \quad\ \text{"}\mathit{congue}\text{"}, \text{"}\mathit{euismod}\text{"}, \text{"}\mathit{non}\text{"}, \text{"}\mathit{mi}\text{"}\ \rangle\end{array}$$

The rule named *count* consumes a string if it is composed of at least two characters, and introduces an integer representing its length into the solution. The *aggregate* rule consumes two integers to produce their sum. By its repeated execution, this rule aggregates the sums to produce the final number. At runtime, these rules are executed repeatedly and concurrently, the first one producing inputs for the second one. While the result of the computation is deterministic, the order of its execution is not. Only the mutual exclusion of reactions by the atomic capture of the reactants is implicitly required by the paradigm.

A possible execution is the following. Let us consider, *arbitrarily*, that the first rule is applied on the first three strings as represented above, and on the last one. The state of the multiset is then the following: $\langle\ \text{"}\mathit{varius}\text{"}, \text{"}\mathit{a}\text{"}, \text{"}\mathit{semper}\text{"}, \text{"}\mathit{congue}\text{"}, \text{"}\mathit{euismod}\text{"}, \text{"}\mathit{non}\text{"}, 8, 6, 5, 2\ \rangle$.

Then, let us assume, still arbitrarily, that the *aggregate* rule is triggered three times on the previously introduced integers, producing their sum. Meanwhile, concurrently, the remaining strings are scanned by the *count* rule. The multiset is then: $\langle\ 6, \text{"}\mathit{a}\text{"}, 6, 6, 7, 3, 2, 21\ \rangle$. With the repeated application of the *aggregate* rule, the inertia is reached ("*a*" satisfies neither of the two rules' conditions but could be removed with a different rule): $\langle\ \text{"}\mathit{a}\text{"}, 51\ \rangle$.

It is important to notice that the atomic capture is a fundamental condition. Let us simply assume that the same string is captured by different nodes running the *count* rule in parallel, then the count for a word may appear more than once in the solution, which would obviously lead to an incorrect result.

In the higher-order version of the chemical programming model [6] any entity taking part in the computation is represented as a molecule (including rules), which unleashes an uncommonly high expressiveness, able to naturally deal with a wide variety of coordination patterns encountered in large scale platforms [3]. However, these works remained mostly conceptual until now.

2.2 System Model

We consider a distributed system $\mathbb{DS}$ consisting of n machines which communicate by message passing. They are interconnected in such way that a message sent from a machine can be delivered, in a finite amount of time, to any other node in $\mathbb{DS}$. At large scale, this can be achieved by relying on P2P systems, more specifically ones employing distributed hash table (DHT) communication protocols [7,8]. They allow us to focus on the atomic capture of molecules without having to worry about the underlying communication.

Data and Rules Dissemination. In the following, we assume data and rules have already been dispatched to nodes. Note that any DHT algorithm or network topology may be used for this purpose. Even if the data and rules are initially held by a single external application, it can contact a node in the DHT and transfer it the chemical solution to be executed. The node which received the data scatters the molecules across the overlay according to the DHT's hash function. Molecules are routed concurrently according to DHT's routing scheme. The dissemination of rules can follow a similar pattern, or be broadcast into the network. The only difference is that rules can be replicated on several nodes to satisfy an increased level of parallelism. A more accurate discussion of the rules' distribution falls out of the scope of this paper. In the following, we simply assume every rule of the program is present on at least one node in the system.

Discovery Protocol. In order for the reaction to happen, a suitable combination of molecules has to be found. While the details of this aspect are also abstracted out in the following, it deserves to be preliminarily discussed. The basic *lookup* mechanism offered by DHTs allows the retrieval of an object according to its (unique) identifier. Unlike the *exact match* functionality provided by DHTs, we require nodes to be able to find *some* molecule satisfying a pattern (*e.g.*, one *integer*) and condition (*e.g.*, *greater than* 3), as stated in Section 2.1. This can be achieved by the support of range queries on top of the overlay network, *i.e.*, mechanisms to find some (at least one) molecules falling within a range, provided the molecules can be totally ordered on some (possibly complex, multi-dimensional) criterion [9]. This mechanism can be easily extended to support patterns and conditions involving several molecules. For instance, when trying to capture two molecules ordered in some specific ways, a *rule translator* — a unit which constructs the range query —, based on the given rule and the first molecule obtained, constructs the range query to be dispatched to the DHT. If matching molecules are found, the capture protocol will be triggered.

Fault tolerance. DHT systems inherently provide a fault-tolerant communication mechanism. If nodes crash, leave or join, the communication pattern will be preserved. On top of that, in this paper we assume there exists a higher-level resilience mechanism which prevents loss of molecules, such as state machine replication [10,11]. Each node replicates its complete state — the molecules and its current actions — across k neighbouring nodes. Thus, in case of its failure, one of its neighbours is able to assume its responsibilities and continue the computation.

3 Protocol

Here, the protocol in charge of the atomic capture of molecules is discussed. The protocol can run in two modes, based on two different sub-protocols: an *optimistic* and a *pessimistic* one. The former is a simplified sub-protocol which is employed while the ratio between actual and possible reactions is kept high. When

this rate drops below a certain threshold, the latter, pessimistic sub-protocol is activated. While being the heavier of the two in terms of network traffic, this sub-protocol ensures the liveness of the protocol, even when an elevated number of nodes in the system compete for the same molecules.

3.1 Pessimistic Sub-protocol

Based on the three-phase commit protocol [12], this sub-protocol ensures that at least one node wanting to execute a reaction will succeed. Molecule fetching is done in three phases — the *query*, *commitment* and *fetch* phases — and involves at least two nodes — the node requesting the molecules, called requester, and the nodes holding the molecules, called holders. Algorithms 3.1 and 3.2 represent the code run on these two entities, respectively, while Figure 1 delivers the time diagram of molecule fetching.

When molecules suitable for a reaction have been found (line 1 in Algorithm 3.1), the query phase begins (line 10). The requester sends *QUERY* messages asynchronously to all of the holders to inform them it is interested in the molecule. Depending on their local states, each of the holders evaluates separately the received message (lines 1—13 in Algorithm 3.2) and replies with one of the following messages: *RESP_OK* (the requested molecule is available), *RESP_REMOVED* (the requested molecule no longer exists) and *RESP_TAKEN* (the molecule has already been promised to another node). Unless it received only *RESP_OK* messages, the requester aborts the fetch and issues *GIVE_UP* messages to holders, informing them it no longer intends to fetch their molecules (line 14 in Algorithm 3.1).

Following the query phase is the commitment phase, when the requester tries to secure its position by asking the guarantee from the holders it will be able to fetch the molecules (line 19 in Algorithm 3.1). It does so using *COMMITMENT* messages. Upon its receipt, each holder sorts all of the requests received during the query phase (line 14 in Algorithm 3.2) according to the conflict resolution policy (described below). Holders reply, once again, with *RESP_OK*, *RESP_REMOVED* or *RESP_TAKEN* messages. A *RESP_OK* response represents a holder's commitment to deliver its molecule in the last phase. Thus, subsequent *QUERY* and *COMMITMENT* requests from other nodes will be resolved with a *RESP_TAKEN* message. Naturally, if a requester does not receive only *RESP_OK* responses to its *COMMITMENT* requests, it aborts the fetch with *GIVE_UP* messages.

Finally, in the fetch phase, the requester issues *FETCH* messages, upon which holders transmit it the requested molecules using *RESP_MOLECULE* messages. From this point on, holders issue *RESP_REMOVED* messages to nodes requesting the molecule.

Conflict Resolution. Each of the holders individually decides to which requester a molecule will be given. Since at least one requester needs to be able to complete its combination of molecules, all holders apply the same conflict resolution

scheme (lines 20—27 in Algorithm 3.2). We here detail a dynamic and load-balancing based scheme: each of the messages sent by requesters contains two fields — the requester's id and the number of reactions it has completed thus far. When two or more requesters are competing for the same molecule, holders give priority to the requester with the lowest number of reactions. In case of a dispute, the requester with a lower node identifier gets the molecule.

Algorithm 3.1: Pessimistic Protocol — Requester.

```
1  on event combination found
2      QueryPhase(combination);
3  on event response received
4      if phase = query then
5          QueryPhaseResp(resp_mol);
6      else if phase = commitment then
7          CommitmentPhaseResp(resp_mol);
8      else if phase = fetch then
9          FetchPhaseResp(resp_mol);
10 begin QueryPhase(combination)
11     phase ⇐ query;
12     foreach molecule in combination do
13         dispatch QUERY(molecule);
14 begin QueryPhaseResp(resp_mol)
15     if resp_mol ≠ RESP_OK then
16         Abandon(combination);
17     else if all responses have arrived
       then
18         CommitmentPhase(combination);
19 begin CommitmentPhase(combination)
20     phase ⇐ commitment;
21     foreach molecule in combination do
22         dispatch COMMITMENT(molecule);
23 begin CommitmentPhaseResp(resp_mol)
24     if resp_mol ≠ RESP_OK then
25         Abandon(combination);
26     else if all responses have arrived
       then
27         FetchPhase(combination);
28 begin FetchPhase(combination)
29     phase ⇐ fetch;
30     foreach molecule in combination do
31         dispatch FETCH(molecule);
32 begin FetchPhaseResp(resp_mol)
33     add resp_mol to reaction_args;
34     if all responses have arrived then
35         Reaction(reaction_args);
36 begin Abandon(combination)
37     phase ⇐ none;
38     foreach molecule in combination do
39         dispatch GIVE_UP(molecule);
```

Algorithm 3.2: Pessimistic Protocol — Holder.

```
1  on event message received
2      if message = GIVE_UP then
3          remove sender from molecule.list;
4      else if message.molecule does not
       exist then
5          reply with RESP_REMOVED;
6      else if message = FETCH then
7          clear molecule.list;
8          reply with molecule;
9      else if molecule has a commitment
       then
10         reply with RESP_TAKEN;
11     else if message = QUERY then
12         add sender to molecule.list;
13         reply with RESP_OK;
14     else if message = COMMITMENT
       then
15         SortRequesters(molecule);
16         if molecule.locker = sender then
17             reply with RESP_OK;
18         else
19             reply with RESP_TAKEN;
20 begin SortRequesters(molecule)
21     foreach pair of requesters in
       molecule.list do
22         if req_j.no_r < req_i.no_r then
23             put req_j before req_i;
24             continue;
25         if req_j.id < req_i.id then
26             put req_j before req_i;
27     molecule.locker ⇐ molecule.list (0);
```

3.2 Optimistic Sub-protocol

When the possibility of multiple, concurrent reactions exists, the atomic fetch procedure can be relaxed and simplified by adopting a more optimistic approach. The optimistic sub-protocol requires only two stages — the *fetch* and the *notification* phases. Algorithm 3.3 describes the sub-protocol on the requesters' side, while Algorithm 3.4 describes it on the holders' side. The time diagram of the process of obtaining molecules is depicted in Figure 2.

Once a node has got information about suitable candidates, it immediately starts the fetch phase (line 1 in Algorithm 3.3). It dispatches *FETCH* messages to the appropriate holders. As with the pessimistic sub-protocol, the holder can respond using the three previously described types of messages (*RESP_MOLECULE*, *RESP_TAKEN* and *RESP_REMOVED*) as shown in Algorithm 3.4. One holder that replied with a *RESP_MOLECULE* message, starts replying with *RESP_TAKEN* messages to subsequent requests until the requester either returns the molecule or notifies it a reaction took place.

If the requester acquires all of the molecules, the reaction is subsequently performed, and the requester sends out *REACTION* messages to holders to notify them the molecules are being consumed. This causes holders to reply with *RESP_REMOVED* messages to subsequent requests from other requesters. In case the requester received a *RESP_REMOVED* message, it aborts the reaction by notifying holders with *GIVE_UP* messages, which allows holders to give molecules to others.

Conflict Resolution. Given the fact that a node will most likely execute the optimistic sub-protocol in a highly reactive stage, there is no need for a strict conflict resolution policy. Instead, the node whose request first reaches a holder obtains the desired molecule. However, the optimistic sub-protocol does not ensure that a reaction will be performed in case of conflicts.

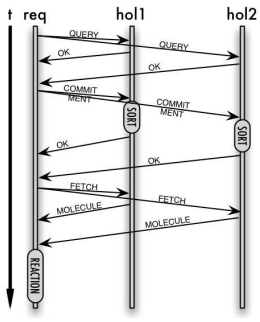

Fig. 1. Pessimistic exchanges.

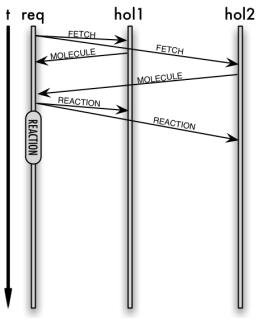

Fig. 2. Optimistic exchanges.

3.3 Sub-protocol Mixing

During its execution, a program typically can pass through two different stages. The first one is a highly reactive stage, which is characterised by a high volume

Algorithm 3.3: Optimistic Protocol — Requester.

```
1  on event combination found
2      foreach molecule in combination do
3          dispatch FETCH(molecule);
4  on event response received
5      if response ≠ RESP_MOLECULE then
6          Abandon(combination);
7          return;
8      add response.molecule to reaction_args;
9      if all responses have arrived then
10         NotifyHolders(combination);
11         Reaction(reaction_args);
12 begin NotifyHolders(combination)
13     foreach molecule in combination do
14         dispatch REACTION(molecule);
15 begin Abandon(combination)
16     foreach molecule in combination do
17         dispatch GIVE_UP(molecule);
```

Algorithm 3.4: Optimistic Algorithm — Holder.

```
1  on event message received
2      if message = GIVE_UP then
3          molecule.state ⇐ free;
4      else if message = REACTION then
5          remove molecule;
6      else if message.molecule does not exist
       then
7          reply with RESP_REMOVED;
8      else if molecule.state = taken then
9          reply with RESP_TAKEN;
10     else
11         molecule.state ⇐ taken;
12         reply with RESP_MOLECULE;
```

of possible concurrent reactions. In such a scenario, the use of the pessimistic sub-protocol would lead to superfluous network traffic, since the probability of a reaction's success is rather high. Thus, the optimistic approach is enough to deal with concurrent accesses to molecules. The second stage is the quiet stage, when there is a relatively small number of possible reactions. Since this entails highly probable conflicts between nodes, the pessimistic sub-protocol needs to be employed in order to ensure the advancement of the system. Thus, the execution environment has to be able to adapt to changes and *switch* to the desired protocol accordingly. Moreover, these protocols have got to be able to coexist in the same environment, as different nodes may act according to different modalities at the same time.

Ideally, the execution environment should be perceived as a whole in which the switch happens unanimously and simultaneously. Obviously, a global view of the reaction potential cannot be maintained. Instead, each node independently decides which protocol to employ for each reaction. The decision is first based on a node's local success rate denoted σ_{local}, computed based on the success history of the last queries the node issued. In order not to base the decision only on its local observations, a node also keeps track of local success rates of other nodes. Each time a node receives a request or a reply message, the sender supplies it with its own current history-based success rate, stored into another list of tunable size. We denote σ the overall success rate, computed as the weighted arithmetic mean of a node's local success rate and the ones collected from other nodes. Finally, the decision as to which protocol to employ depends on the rule a node wishes to execute. More specifically, it is determined by the number of the rule's arguments, since the more molecules the rule needs, the harder it is to assure they will be grabbed: to grab r molecules, a node employs the optimistic

sub-protocol if and only if $\sigma^r \geq s$, where r is the number of arguments the chosen rule has and s is a predefined threshold value. If the inequality is not satisfied, the node employs the pessimistic sub-protocol.

Coexistence. Due to the locality of the switch between protocols, not all participants in the system will perform it in the exact same moment, leading to possible inconsistencies in the system, where some nodes try to grab the same molecules using different protocols. In order to distinguish between *optimistic* and *pessimistic* requests, each requester incorporates a *request type* field into the message being sent. Based on this field, the node holding the conflicting molecules gives priority to nodes employing the more conservative, pessimistic algorithm. Although this decision discourages *optimistic* nodes and sets them back temporarily, it ensures that, in the long run, *eventually* a node will be able to grab the molecules it needs, since pessimism is favoured over optimism.

3.4 Sketch of Proof for Correctness and Liveness

The proposed protocol is a combination of the extensions of two existing protocols presented in [12] and [13]. These two protocols were initially introduced to guarantee resource transactions with only one holder. In our context, a requester can ask for several molecules owned by different holders.

These protocols must guarantee two properties: i) *correctness*: a molecule is used in only one reaction (as we consider that every reaction consumes all of the molecules entering it), and ii) *liveness*: if a node sends a request infinitely often, it will eventually succeed in capturing the molecules, provided the requested molecules are still available.

Correctness Proof. Correctness is easy to prove because both protocols we rely on have been proved to be correct independently. There are two cases of conflict between the two protocols. When an optimistic request arrives before a pessimistic one, the pessimistic request is aborted because the molecule has already been reserved by the optimistic requester. On the other hand, if a pessimistic request arrives first, the optimistic request is aborted in favour of the pessimistic one.

Liveness Proof. To prove the liveness property, we show that: i) if no successful reaction happens in the system, nodes eventually switch to the pessimistic protocol, ii) if several pessimistic requesters are in conflict, at least one reaction is not aborted, and iii) a node cannot see its reactions infinitely aborted.

Initially, and hopefully most of the time, nodes use the optimistic sub-protocol for their requests. In case of a conflict between two optimistic requesters, both requests can easily be aborted. Consider the example where two concurrent requesters try to capture two molecules, A and B. If the first requester succeeds in grabbing A while the second captures B, then the two requests will be aborted.

For the pessimistic protocol, we define a total order based on the number of successfully completed reactions by a node and its id. In case of a conflict, all of the reactions might be aborted except for one — the reaction initiated by the node which comes first as per the total order. Because the total order is based on the number of successful reactions, if a node, in case of an abort, tries again infinitely to request molecules for its reaction, eventually, if the requested molecules are still available, the reaction will take place, given the fact that its position moves up the total ordering when other nodes succeed in executing their reactions.

When a request of a node is aborted, the node decreases its value of σ (see Section 3.3). With each message sent, the node includes the information about its local σ, and collects the values received with each message. If there are many conflicts during a certain period of time, all the more so if there is no successful reaction, the local σ of all of the nodes decreases. This effect leads to a situation where the value of the computed σ^r for all new reactions will be lower than the threshold s, which will force the nodes to use the pessimistic protocol when initiating new requests, which insures the system's liveness.

When presenting algorithms for atomic capture, it is common to study their convergence times. However, any discussion about convergence when dealing with the chemical programming model is not feasible, as convergence itself, and thus the convergence time, is an application-specific property. However, the next section presents an evaluation of the proposed algorithm, and sheds some light on the subject.

4 Evaluation

Our protocol was simulated in order to better capture its performances. We developed a Python-based discrete-time simulator, including a DHT layer performing the random dissemination of a set of molecules over the nodes, on top of which the layer containing the capture protocol itself was built. At this layer, any message issued at step t will be received and processed by the destination node at time $t + 1$. Moreover, each time a capture attempt either led to a reaction, or an abortion, the node tries to fetch another set of r randomly chosen molecules. Finally, on the top layer, a simple chemical application was simulated.

All presented experiments simulate a system of 250 nodes trying to execute a chemical program containing a solution with 15 000 molecules and a straightforward rule which simply consumes two molecules without producing new ones. Such a simple program allows us to concentrate exclusively on evaluating the capture protocol itself, without having to deal with application-specific logic. In the same vein, reactions' duration are assumed negligible. Each simulation was run 50 times and the figures presented below show the values obtained by averaging result data from these runs. Keep in mind that the final steps of the executions shown in the figures represent, due to the effect of averaging, worst-case scenarios obtained during simulation. Simulations were limited to execute at most 500 steps, as later steps are not relevant.

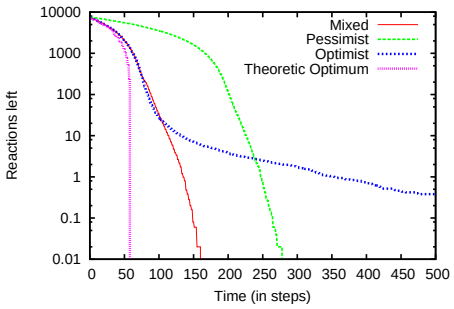

Fig. 3. Performance comparison of the protocol's variants.

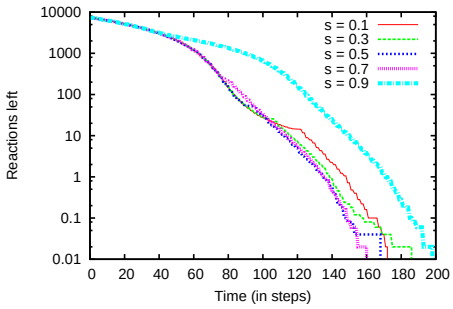

Fig. 4. Execution time for different switch thresholds.

Experiment 1. Firstly we evaluate separately the performance characteristics of both sub-protocols. Figure 3 shows the averaged number of reactions left to execute at each step, until inertia, using only the optimistic mode, only the pessimistic mode, and the complete protocol with switches between protocols (using $\sigma = 0.7$), respectively. Note that a logarithmic scale is used. The figure shows that, using only the *optimistic* protocol, while we can see a strong decline in the number of reactions left at the beginning of the computation, *i.e.*, when a lot of reactions are possible and that thus there are only few conflicts in the requests, it gets harder for nodes to grab molecules when this number declines. In fact, the system is not even able, for most of the executions, to conclude the execution, as a few reactions left are never executed, always generating conflict at fetch time. When the nodes are all *pessimistic*, there is a steady, linear decrease in the number of reactions left, and the system is able to reach the inertia in a reasonable amount of time, thanks to the liveness ensured in this mode. For most steps, the *mixed* curve traces the exact same path as the *optimistic* one, which means that during this period the nodes employ the optimistic sub-protocol. However, at the end, the system is able to quickly finish the execution as an aftermath of switching to the pessimistic protocol. After the switch, it diverges from the optimistic one to mimic the pessimistic curve, exhibiting a benefit of a 42% performance boost compared to the performance of the pessimistic sub-protocol. Finally, the *theoretic optimum* curve represents the minimal amount of steps needed to complete the execution in a centralised system. Comparing it to our protocol, we notice an increase of 166% in the number of steps needed to reach inertia. This is understandable, because there is usually a coordinator in centralised systems with which conflict situations can be circumvented, but it opens the door to serious defaults, such as single-point-of-failure or bottleneck problems.

Experiment 2. Next, we want to assess the impact of the switch threshold s on the overall performance of the system. Figure 4 depicts, in a logarithmic scale, the number of reactions left on each step for different threshold values, varying

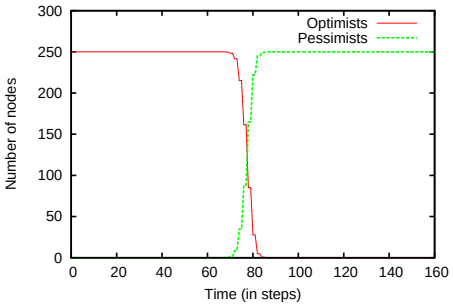

Fig. 5. Number of nodes employing optimistic and pessimistic protocols per step.

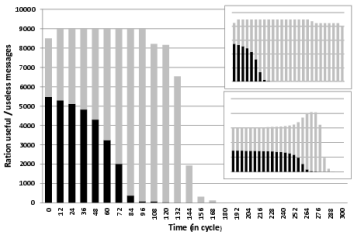

Fig. 6. Number of messages sent per cycle.

from 0.1 to 0.9. As suspected, the curves overlap during most steps, most nodes employing the optimistic sub-protocol. The first curve to diverge is the one where the switch threshold is set very high, to $s = 0.9$. Because the system depicted by that curve did not fully exploit the optimistic sub-protocol, it is the last to finish the execution. Although slightly, the other curves start diverging at different moments, and, thus, complete the execution at different steps. Looking at Figure 4 brings us to the conclusion that, out of the five values tested for the switch threshold, $s = 0.7$ yields the best performance results in this particular scenario. Finding an overall optimal value for s falls out of the scope of this paper.

Experiment 3. Here we examine the properties of the process of switching from one protocol to the other. Figure 5 shows that, at the beginning of the execution, all of the nodes start off grabbing molecules by using the optimistic sub-protocol. The switch happens about half way through the execution. Around that time, *optimistic* nodes can no longer efficiently capture molecules, so they switch to the pessimistic sub-protocol. We observe that, due to exchange of local σ values, nodes in the system reach a global consensus rather quickly — for a system with 250 nodes, at most 15 steps are needed for all of the nodes to switch to the pessimistic protocol.

Experiment 4. Finally, we investigate the communication costs involved in the process. Figure 6 depicts the number of messages sent per cycle (where one cycle comprises 12 simulation steps), classified into two categories: useful messages (ones which led to a reaction, in black) and useless messages (ones which did not induce a reaction, in grey). We note that the protocol takes over the best properties of both of its sub-protocols. Firstly, it takes over the elevated percentage of useful messages of the optimistic sub-protocol. After the switch, the pessimistic protocol kicks in, bringing with it a decrease in the total number of messages. When compared to the communication costs of each of the sub-protocols separately (both depicted on the right-hand side of Figure 6), we see that switching from one protocol to the other reduces network traffic and improves scalability.

5 Related Works

The chemical paradigm was originally conceived for programs which need to be executed on parallel machines. The pioneering work of Banâtre *et al.* [4] provides two conceptual approaches to the implementation problem, in both of which each processor of a parallel machine holds a molecule and compares it with the molecules of all the other processors. A slightly different approach was proposed in the work of Linpeng *et al.* [14], where a program is executed by placing molecules on a strip, and then folding them over after each vertical comparison. Recently, Lin *et al.* developed a parser of GAMMA programs for their execution on a cluster exploiting GPU computing power [15]. All works mentioned exhibit significant speed-up properties, but the platforms experimented are rather restricted.

Mutual exclusion and resource allocation algorithms have been studied extensively. Nevertheless, most research focuses on sharing one specific resource, or critical section, amongst many processes [16,17]. A basic solution for the *k-out of-M* problem was given by Raynal [18]. This early work is a static permission-based algorithm in which only the number of a predefined set of resources varies from node to node. In addition, the solution supposes a global knowledge of the system. On the other hand, an execution environment for chemical programs is a dynamic system in which nodes need to obtain different molecules, which can be thought of as resources, at different times.

The three-phase commit protocol was originally proposed as a crash recovery protocol for distributed database systems [12]. Although, in its essence similar to the three-phase commit protocol, the goal of the optimistic sub-protocol proposed in this paper is to secure the liveness of the system by ensuring that at least one node will be able to complete its reaction.

6 Conclusion

While chemical metaphors are gaining attention in the modelling of autonomous service coordination, the actual deployment of programs following the chemical programming model over distributed platforms is a widely open problem. In this paper, we have described a new protocol to capture several molecules atomically in an evolving multiset of objects distributed on top of a large-scale platform. By dynamically switching from one sub-protocol to the other, our protocol fully exploits their good properties (the low communication overhead and speed of the optimistic protocol, when the density of reactants is high, and the liveness guarantee of the pessimistic protocol, when this density drops), without suffering from their drawbacks. These features are illustrated by simulation.

This protocol is part of an ambitious work which aims at building a distributed autonomic platform providing all the features required to execute chemical programs. This work is quite interesting in that it revisits classical problems in distributed systems, but with the large scale requirements, as well as the specificities of the chemical model, in mind. In this way, this paper tackles the mutual exclusion: in our context the liveness property is a system property while, more traditionally, liveness is a process' property. Among the directions planned for this work, we will refine the execution model, to, for instance, balance the load of reactions among the nodes of the platform. On the practical side, we plan to use these algorithms to actually leverage the expressiveness of the chemical paradigm for a workflow management system such as defined in [19].

References

1. P. Dittrich, J. Ziegler, and W. Banzhaf, "Artificial chemistries – a Review," *Artificial Life*, vol. 7, pp. 225–275, June 2001.
2. M. Viroli and F. Zambonelli, "A Biochemical Approach to Adaptive Service Ecosystems," *Information Sciences*, 2009.
3. J.-P. Banâtre and T. Priol, "Chemical Programming of Future Service-oriented Architectures," *Journal of Software*, vol. 4, 2009.

4. J.-P. Banâtre, A. Coutant, and D. Le Metayer, "A parallel Machine for Multiset Transformation and its Programming Style," *Future Gener. Comput. Syst.*, vol. 4, pp. 133–144, September 1988.
5. L. Lamport, "Ti clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, pp. 558–565, July 1978.
6. J.-P. Banâtre, P. Fradet, and Y. Radenac, "Generalised Multisets for Chemical Programming," *Mathematical Structures in Computer Science*, vol. 16, 2006.
7. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *SIGCOMM*, pp. 149–160, 2001.
8. A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Middleware*, 2001.
9. C. Schmidt and M. Parashar, "Squid: Enabling search in dht-based systems," *J. Parallel Distrib. Comput.*, vol. 68, no. 7, pp. 962–975, 2008.
10. F. B. Schneider, "Implementing Fault-Tolerant Services Using the State Machine Approach: a Tutorial," *ACM Comput. Surv.*, vol. 22, 1990.
11. N. A. Lynch, D. Malkhi, and D. Ratajczak, "Atomic data access in distributed hash tables," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, (London, UK), pp. 295–305, Springer-Verlag, 2002.
12. D. Skeen and M. Stonebraker, "A Formal Model of Crash Recovery in a Distributed System," *IEEE Transactions on Software Engineering*, vol. SE-9, no. 3, 1983.
13. B. W. Lampson and H. E. Sturgis, "Crash recovery in a distributed data storage system," 1979.
14. L. Huang, W. Tong, W. Kam, and Y. Sun, "Implementation of gamma on a massively parallel computer," *Journal of Computer Science and Technology*, vol. 12, pp. 29–39, 1997.
15. H. Lin, J. Kemp, and P. Gilbert, "Computing Gamma Calculus on Computer Cluster," *IJTD*, vol. 1, no. 4, pp. 42–52, 2010.
16. B. A. Sanders, "The information structure of distributed mutual exclusion algorithms," *ACM Transactions on Computer Systems*, vol. 5, no. 3, pp. 284–299, 1987.
17. K. M. Chandy and J. Misra, "The drinking philosophers problem," *ACM Transactions on Programming Languages and Systems*, vol. 6, no. 4, pp. 632–646, 1984.
18. M. Raynal, "A Distributed Solution to the k-out of-M Resources Allocation Problem," *Advances in Computing and Information — ICCI'91*, pp. 599–609, 1991.
19. H. Fernandez, T. Priol, and C. Tedeschi, "Decentralized approach for execution of composite web services using the chemical paradigm," in *ICWS*, pp. 139–146, 2010.